REMARKS

In the May 21, 2008, Office Action, the United States Patent and Trademark Office ("Office") again rejected Claims 1-54 under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of U.S. Patent No. 6,701,383 ("Wason et al."), and further in view of the teachings of U.S. Patent No. 7,171,448 ("Danielsen et al."). Although applicants disagree with the Office, for the sake of advancing the prosecution of the above-identified patent application, applicants have amended Claims 1 and 39 to help the Office appreciate the differences between the claimed subject matter and the teachings and suggestions of the applied and recited references.

Applicants are unable to find, and the Office has failed to show, where the cited and applied references teach or suggest the claimed subject matter. For example, applicants are unable to find where the cited and applied references teach or suggest "a web browser and not a media player configured to provide timing representations to media players," as recited by Claims 1 and 39, albeit in a different manner. The Office withdrew the previous notion that this recitation is somehow met by Wason et al. at Col. 1, lines 29-31, which is located in his Background of the Invention, but now the Office has indicated that such a claimed feature can be found somewhere else in Wason et al., namely at Col. 2, lines 63-67, which applicants represent here in full:

> Some of the features of this invention are listed below: 1. Secure e-commerce, full interactive experiences including text and voice chat, games, graphics, animations, and advertising can be integrated and synchronized with streaming multimedia such as video and audio; 2. It can be used to build a content framework that insulates content and plug-in developers from details and differences in hardware platforms, so that the same content or plug-in can run on desktop platforms (PC, Macintosh, Solaris, Linux), Televisions (GI, SA), or other kinds of devices (AOL-TV, screen phones); 3. It can be used to build a content framework that insulates content and plug-in developers from specifics of, or differences in, software platforms, so that the same content or plug-in can run on Microsoft NetShow™, RealNetworks RealPlayer™, Apple Quicktime™,

> Sun Java™ Media Framework or any other media system; 4. It can run without a network or on different types of networks, such as wireless, Intranets, or the Internet; 5. It can be used to synchronize arbitrary data types, including text, chat, graphics, video, audio, and active content like Java™, JavaScript, or Flash . . . .

The fact that the system of Wason et al. can run without a network or on an Internet network does not mean that Wason et al. discloses "a web browser and not a media player configured to provide timing representations to media players," as recited by Claims 1 and 39, albeit in a different manner. In other words, one with ordinary skill in the art would understand the differences between an Internet network and a Web browser. A Web browser is not an Internet network. An Internet network is not a Web browser. The Office also points to Wason et al. at Col. 3, lines 1-3, which discloses as follows:

> 6. The framework-independent layer can be implemented using different languages, including Java™, HTML, XML, JavaScript, VBScript, or other languages; 7. When used for video synchronization, it can be used to synchronize arbitrary datatypes with different spatial objects in video, and with different temporal objects in video.

The fact that the framework of Wason et al. can be implemented in Java or HTML fails to disclose "a web browser and not a media player configured to provide timing representations to media players," as recited by Claims 1 and 39, albeit in a different manner. In other words, one with ordinary skill in the art would understand the differences between a computer language, such as Java and HTML, and a Web browser. A Web browser is not a computer language. A computer language is not a Web browser.

The focus of the Office on the Web browser misses the point. The point is that there has to be a Web browser and not a media player that provides timing representations to media players. Wason et al. discloses a multimedia player coordinating with an abstraction layer to provide synchronization. This is one difference, but there are many others.

There is a continued confusion among the words "framework," plug-in," and "multimedia player." Wason et al. uses the word "framework" interchangeably with the word "multimedia player." For example, Wason et al. explains at Col. 1, line 27, that "[m]ultimedia player frameworks have become widespread." As a second example, Wason et al. explains at Col. 1, lines 32-36, that "[t]hese are frameworks such as RealNetworks, Inc.'s RealPlayer™ G2 family," and so on. As a third example, Wason et al. explains that "[m]ost of these frameworks are extensible by means of plug-ins discussed below," and so on. See Col. 1, lines 37-39. In other words, a "plug-in" is not a framework but that which may be connected with a framework to extend the framework. Wason et al. explains what his invention is about at Col. 1, lines 19-24:

> This invention relates to the field of software plug-ins for multimedia file players and for other applications supporting ordered data flow files. More precisely, this invention defines a new field of software that allows plug-ins and content to be insulated from differences in underlying platforms and frameworks.

In other words, Wason et al. has nothing to do with the Web browser but focuses entirely on frameworks and plug-ins. In contrast, applicants have explained as follows:

> [0002] Conventional Internet browsers were designed mainly as text layout engines. Such browsers, therefore, are typically very limited in the ways in which they deliver multimedia content. As broadband Internet access becomes more widely available, however, multimedia playback of content, including, but not limited to video content and audio content, will become an increasingly important feature that an Internet browser should provide.
>
> [0003] Conventionally, upon encountering an embedded multimedia object, a browser merely provides a rendering area and does not stay involved with communicating timing information to a media player or passing synchronization information between a media player and other types of content. Instead, with respect to timing, the media player is essentially autonomous once it has been instantiated, and provided a rendering area, from a browser.

The multimedia player of Wason et al. remains essentially autonomous from a Web browser, as explained at Col. 2, lines 27-41:

> This invention is an abstraction layer providing a uniform interface between a framework and one or more plug-ins. In the preferred embodiment, the invention is a Synchronization Abstraction Layer (SAL) abstracting time-based frameworks into a common synchronization interface. The SAL synchronizes itself and other plug-ins to a time-line of the underlying framework--and it does that independently of the underlying framework. In other words, the plug-ins interact with the underlying framework through the SAL, rather than directly. Typically, the SAL is implemented on top of the synchronization of the Application Programming Interfaces (API's) provided by the underlying frameworks. It has at least one point of coupling with the underlying framework: a method for providing the SAL with the current time.

Wason et al. provides a very specific and simple to understand example to show that Wason et al. is focusing on his framework or multimedia player and plug-ins that extend the framework of the multimedia player. That example is provided here in full in hope that the Office would review it and applicants have even underlined salient portions:

> FIG. 3 illustrates a high level diagram of a more specific example of the preferred embodiment. Here, the SAL is used to plug-in a table of contents ("TOC") extension module into the RealPlayer™. The TOC plug-in displays a hierarchical view of the table of contents of a video presentation described in the table of contents. In the hierarchy, entries in the table are highlighted during the video presentation of their associated video portions. A user can randomly access various portions of the video by clicking on the corresponding entries in the TOC.
>
> In this configuration RealPlayer™ 301 has three extension modules: SAL (RealJava) 310, RealVideo 302, and RealText 303. (Hereinafter we denominate by "rj" or "RealJava" all Java™ objects compatible with the G2 architecture.) For clarity, SAL 310 is expanded within the dotted oval into its conceptual components--File Format Filter 311 and Renderer 312. This conceptual separation results from the G2 architecture constraints of the RealPlayer™; in a SAL adaptation compatible with another media player, such conceptual separation may be inappropriate.

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue
Suite 2800
Seattle, Washington 98101
206.682.8100

When a user invokes the RealPlayer™ application and the TOC extension module, RealPlayer™ 301 accesses Web Server 320 over Network 330 to retrieve "toc.smi," a synchronized multimedia integration language ("SMIL") descriptor file defining a set of synchronized plug-ins to run simultaneously in the RealPlayer™ environment. This file contains a reference to "toc.rj," an extensible markup language ("xml") descriptor file for the table of contents extension module compatible with the RealPlayer™. Based on the MIME type of the toc.rj file, RealPlayer™ 301 instantiates SAL 310 and routes the toc.rj file to it. SAL 310 includes File Format Filter 311 and Renderer 312, each implemented using Java™ code linked with native platform C++ code.

File Format Filter 311 parses the toc.rj file and hands it over to Renderer 312 through a standard protocol defined by the RealPlayer™. The parsed data contains following elements: (1) name of the Java™ class to instantiate; (2) location from which the Java™ class can be retrieved; and (3) any other parameters specific to the extension module. A sample toc.rj file appears in FIG. 4. The first line (starting with classid) specifies the Java™ class names; the next two lines (codebar and archive) specify a URL location and names of Java™ class files; the following line assigns the width and the height of the window for the table of contents; sync is the period in milliseconds between synchronizing events, i.e., between timing calls from the RealPlayer™ to the SAL; duration refers to total length of the time-line of the presented event; and, lastly, param is the parameter that points to the URL with the toc.xml file, the XML file descriptor of the specific table of contents.

Renderer 312 instantiates JVM (Java™ Virtual Machine) 313 needed to run the Java™ code, and also retrieves ".jar" files (packaged Java™ class files) over Network 330. Note that the URL for the ".jar" files is available from the parsed toc.rj file discussed above. Renderer also instantiates an instance of RealTOC object 314 and hands to it the URL for toc.xml, the XML file describing the specific table of contents object associated with the multimedia presentation. This file includes the nodes and sub-nodes within hierarchical structure of the specific table of contents; it also includes the "begin" and "end" times corresponding to each node and sub-node. RealTOC retrieves the toc.xml file, parses it, and builds the specific table of contents tree structure. A sample toc.xml file appears in FIG. 5.

At the beginning of the multimedia presentation, RealVideo object 302 creates a video window for rendering video, while RealTOC creates a TOC window for rendering the table of contents. Both windows are rendered within the larger RealPlayer™ window. As the time-line

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue
Suite 2800
Seattle, Washington 98101
206.682.8100

> increases during the presentation, RealPlayer™ 301 periodically calls SAL 310 with the current time; SAL 310 passes the current time to JVM 313, which in turn notifies RealTOC 314; RealTOC 314 highlights the appropriate node on the table of contents tree rendered in the TOC window. A similar process can be constructed with the SAL keeping current time and calling the RealPlayer™ with time updates.
>
> When the user clicks on a particular heading within the table, RealTOC 314 sends the "begin" time associated with the heading to the "seek" function of RealPlayer™ 301 through JVM 313 and SAL 310; RealPlayer™ 301 notifies all synchronized media servers (not shown) of the new "begin" time, waits for the all the media servers to synchronize to the new time, then updates its internal current time and sends the new current time to its extension modules, i.e., RealVideo 302 and SAL 310 (and RealText 103, if applicable); SAL 310 updates the current time of RealTOC 314 through JVM module 313. Importantly, both RealTOC 314 and RealVideo 302 are both synchronized to the current time of RealPlayer™ 301. Thus, after RealTOC 314 invokes the seek function requesting a new time, its current time does not skip to the new time until RealPlayer™ 301 notifies RealTOC 314 of the change in current time. If the seek function in RealPlayer™ 301 is disabled, as it should be for live video, the time line continues uninterrupted.

Wason et al. uses the RealPlayer™ multimedia player in coordination with the SAL to synchronize. No where is there any involvement of the Web browser to provide timing representation to multiple media players. As applicants have explained above, the problem is as follows:

> [0003] Conventionally, upon encountering an embedded multimedia object, a browser merely provides a rendering area and does not stay involved with communicating timing information to a media player or passing synchronization information between a media player and other types of content. Instead, with respect to timing, the media player is essentially autonomous once it has been instantiated, and provided a rendering area, from a browser.

The Office has sought to combine Wason et al. and Danielsen et al., which combination applicants specifically deny. Wason et al. has defects which are not cured by Danielsen et al. and, consequently, there is no reason to combine them. Thus, the Office has failed to state a

prima facie case of obviousness. Because the Office has failed to state a *prima facie* case of obviousness, the rejections should be withdrawn. Independent Claims 1 and 39 are clearly patentably distinguishable over the cited and applied references. Claims 2-38 and 40-54 are allowable because they depend from allowable independent claims and because of the additional limitations added by those claims. Consequently, reconsideration and allowance of Claims 1-54 is respectfully requested.

Respectfully submitted,

CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC

D.C. Peter Chu
Registration No. 41,676
Direct Dial No. 206.695.1636

DPC:jlg